17. Werken met bestanden

We gebruiken het flash-geheugen van een controllerbordje voor het opslaan van programmabestanden. Main.py en de librarybestanden horen daar ook. Het flash is groot genoeg om er ook gegevensbestanden te plaatsen. Een voorbeeld is een logger. De meetwaarden van sensoren slaan we op in een bestand dat we later analyseren.

Bestanden: lezen en schrijven

Niet voor SAMD2 , wel voor de ESP's en RP2040

Voor we een bestand gebruiken, moeten we het openen. We gebruiken **klasse open**. De constructor is:

F = Open(*bestand*, *mode*)

Bestand, de bestandsnaam is een string. Mode geeft aan hoe we het bestand gebruiken.

Mode	
"r"	Open een bestaand bestand om gegevens te lezen.
"r+"	Open een bestaand bestand om te lezen en te schrijven. De bestandscursor staat in het begin.
"w"	Maak een nieuw bestand en open het om te schrijven. Als bestand reeds bestaat, wordt het overschreven.
"w+"	Maak een nieuw bestand en open het om te schrijven en te lezen .Als bestand reeds bestaat, wordt het overschreven.
"a"	Open een bestand om bij te voegen. Als bestand niet bestaat, wordt het aangemaakt. Als bestand bestaat, dan staat de bestandscursor aan het einde.
"a+"	Open een bestand om bij te voegen en te lezen. Als bestand niet bestaat, wordt die aangemaakt. Als bestand bestaat, dan staat de bestandscursor aan het einde.

We schrijven naar een bestand met methode write() en we lezen uit een bestand met methode read(). Na gebruik moet je het bestand sluiten met methode close(). Enkele voorbeelden:

We maken een nieuw bestand en we schrijven er een regel tekst in.

```
file = open("test.txt", "w")
file.write("regel 1")
file.close
```

We schrijven bijkomende regels in het bestand. Met \n, het nieuwe-regel-teken laten we tekst een nieuwe regel beginnen in het bestand.

```
file = open("test.txt", "a")
file.write("\ntweede regel")
```

```
file.write("\nnog een regel")
file.close
```

We lezen het bestand.

```
file = open("test.txt", "r")
str1 = file.read()
print(str1)
file.close
```

De drie voorbeelden hierboven werken niet op een SAMD21.

Enkele bestandsbewerkingen

Voor bestandsbewerkingen gebruiken we klasse os. Enkele veel gebruikte functies zijn:

- **listdir()** geeft een lijst van de bestanden
- **mkdir()** maakt een nieuwe directory (ma, folder) aan
- **chdir()** verandert de actieve directory
- **mdir()** verwijdert een directory
- **getcwd()** geeft de huidige directory
- **rename()** geeft een bestand of directory een nieuwe naam
- **remove()** verwijdert een bestand
- **stat()** geeft de status van een bestand of directory

De figuur hieronder toont enkele voorbeelden.

```
>>> import os
>>> os.listdir()
['DS1302.py', 'i2c_lcd.py', 'i2c_lcd_backlight.py'...
>>> os.getcwd()
'/'
>>> os.mkdir("teksten")
>>> os.chdir("teksten")
>>> os.getcwd()
'/teksten'
>>> os.chdir("/")
>>> os.stat("DS1302.py")
(32768, 0, 0, 0, 0, 0, 3988, 1614102800, 161410280...
```

Figuur 102: klasse os: het bestandssysteem

Bestandsfouten

Bij het uitlezen van een bestand kan toch iets fout gaan: het bestand bestaat niet, MicroPython geeft dan foutmelding *OSError: 2*. Met een try:-constructie gaan we dat opvangen.

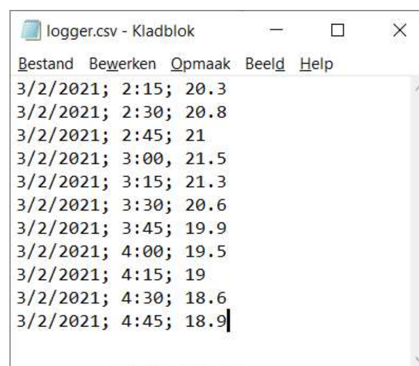
```
try:
```

```
    file = open("tekst.txt", "r")
except OSError:
    print("bestand niet gevonden")
else:
    str1 = file.read()
    print(str1)
    file.close
```

Met try: proberen we het bestand te openen. Als de OSError zich voordoet, dan drukken we af dat het bestand niet gevonden is. In het andere geval, lezen we alle regels van het bestand en drukken ze af.

Toepassing: een temperatuurlogger

De temperatuurlogger zal op vaste tijdstippen de omgevingstemperatuur meten en die met datum en uur bewaren in een bestand. Dat bestand kan je dan met Thonny opladen en bewaren op je PC voor een verdere analyse. We gebruiken daarvoor een CSV-bestand (Comma Separated Values). Dat kunnen we opladen in Excel.



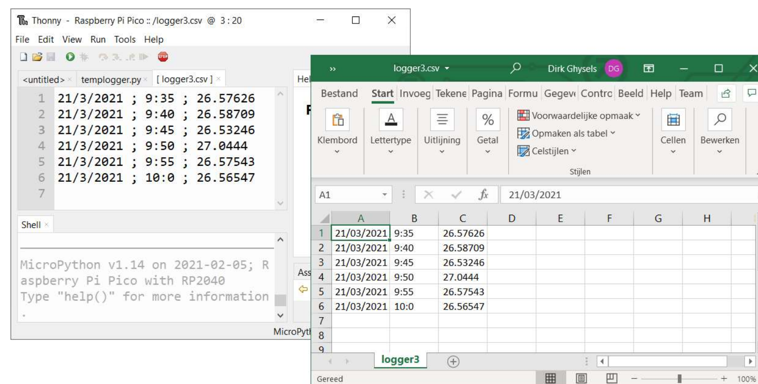
Figuur 103: CSV-bestand

In een CSV bestand hebben alle lijnen dezelfde vorm. In het voorbeeld bestaat elke lijn uit een datum, het uur en een getal. Tussen de elementen staan punt-komma's als scheidingsteken.

```
#-----------------------------#
#  templogger.py              #
#                             #
#  21 maart 2021              #
#  Dirk Ghysels               #
#-----------------------------#
from machine import Pin, Timer
import utime
min0 = 0
uur = 0
adc = machine.ADC(4)
ti = 5  # tijd tussen 2 metingen in minuten
```

```
while(True):
    time = utime.localtime()
    uur = time[3]
    min = time[4]
    dag = time[2]
    maand = time[1]
    jaar = time[0]
    if min!=min0:
        min0 = min
        if min % ti == 0:
            R = adc.read_u16()
            S = R*3.3/65535
            temp = 27 - (S-0.706)/0.001721
            file = open("logger3.csv", "a")
            lijn = str(dag)+"/"+str(maand)+"/"+str(jaar)
            lijn += " ; "+str(uur)+":"+str(min)+" ; "
            lijn += str(temp)
            file.write(lijn+"\n")
            file.close
    utime.sleep(5)
```

Het programma is een samenraapsel van stukken uit programma’s van vorige hoofdstukken. Het programma is opgebouwd rond een klokprogramma. Elke minuut gaat het na of het aantal minuten deelbaar is door de tijd tussen de metingen (min%ti == O). Indien ja, lezen we de temperatuur met de interne sensor en schrijven een regel in het bestand. Die regel bestaat uit datum, uur en temperatuur.Het bestand staat in de flash van de controller. Met Thonny kan je het openen, en wegschrijven naar de schijf van je PC. Het is een CSV-bestand, Excel kan het lezen.



Figuur 104: logbestand in Thonny en in Excel